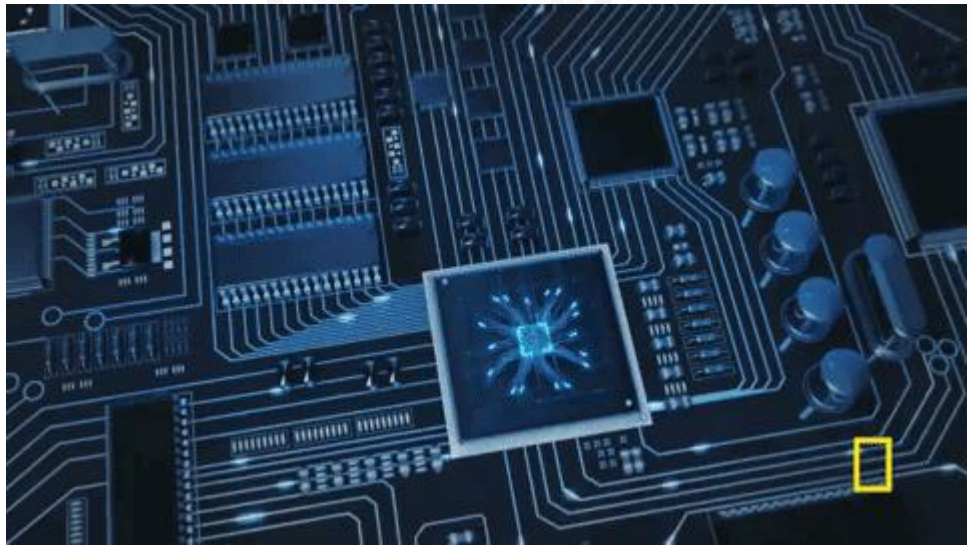
ATME College of Engineering

Department of EEE
Emitting Elite Energy

BEE613B
Embedded Systems Design
Module-3: Hardware Software Co design and Program Modelling



Presented by,

Mr.Shreeshayana R

Assistant Professor

Electrical and Electronics Engineering

ATME College of Engineering, Mysuru

ATME College of Engineering

Department of EEE
Emitting Elite Energy

Contents

3.1 Hardware Software Co design and Program Modelling : Fundamental issues in Hardware Software Co-design,
3.2 Computational models in Embedded System Design (Chapter 7 – Text 1: 7.1, 7.2)
3.3 Embedded Hardware Design and Development: Analog Electronic Components,
3.4 Embedded Hardware Design and Development :Digital Electronic Components,
3.5 VLSI & Integrated Circuit Design,
3.6 Electronic Design Automation Tools

Learning Objectives

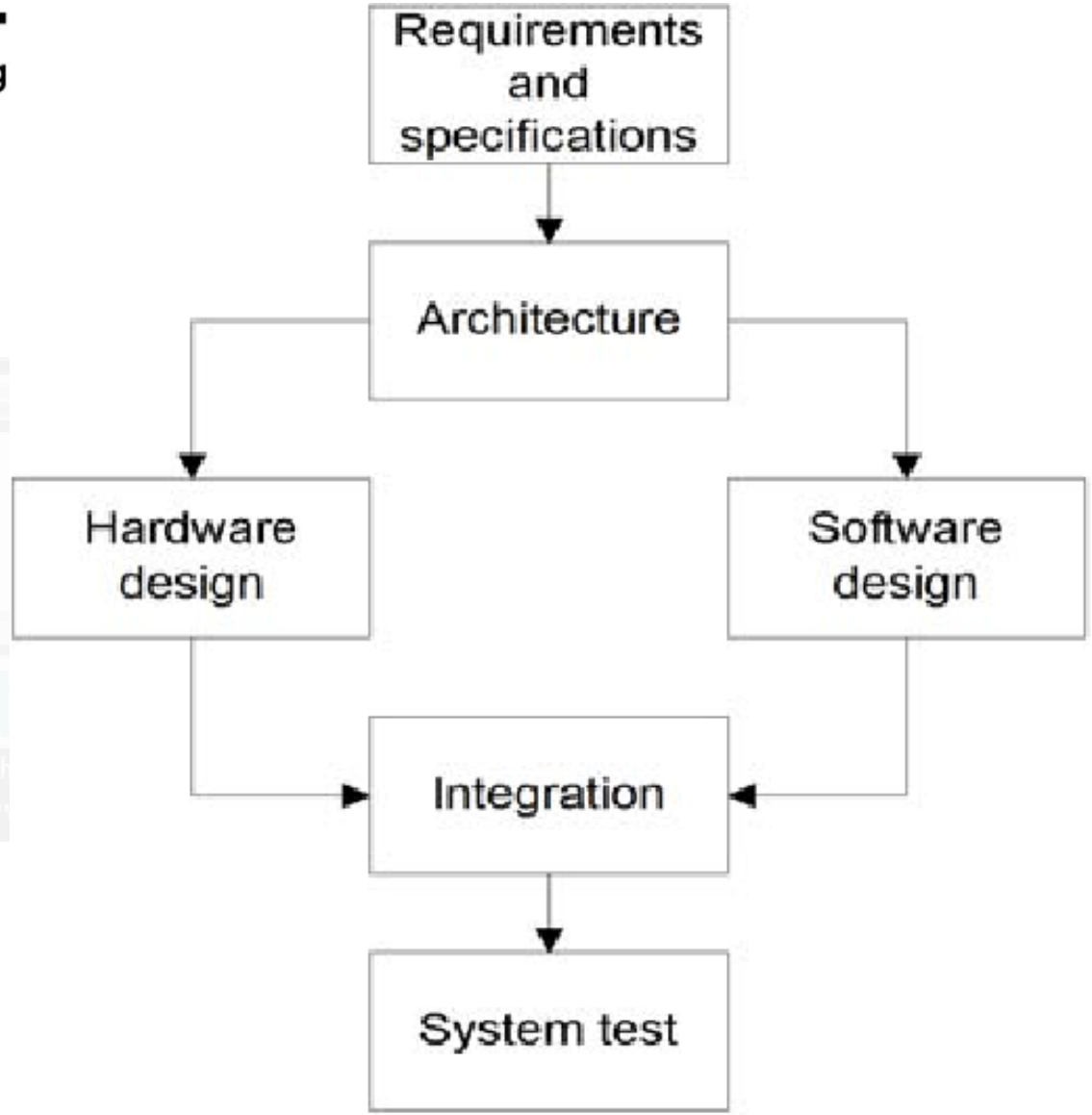
1.Understand Hardware-Software Co-Design: Analyze key issues and architectural trade-offs.

2.Explore Computational Models: Familiarize with models such as FSM, DFG, and CDFG.

3.Understand Embedded Hardware Design: Study the role of analog and digital components.

4.Learn VLSI Design Concepts: Understand IC types, design steps, and methodologies.

5.Introduction to EDA Tools: Study modern tools like OrCAD, Eagle, and Prote


Requirements and specifications
Architecture
Hardware design
Software design
Integration
System test

3.1 Hardware Software Co-Design and Program Modelling

Fundamental Issues in Hardware-Software Co-Design:

1. Selecting the Model:

1. Specification Stage: Focus on **functionality**.
2. Implementation Stage: Focus on system **structure**.

- **Phase 1: Functional Model**

Focuses on describing *what* the system should do **without worrying about implementation**.

- **Phase 2: Structural Model**

As you move towards implementation, the model shifts to describing *how* the system components interact and are structured.

- **Phase 3: Transitioning Models**

As design evolves, switching between models is necessary, but the complexity increases.

•**Phase 1: Functional Model**

Focuses on describing *what* the system should do **without worrying about implementation**.

•**Phase 2: Structural Model**

As you move towards implementation, the model shifts to **describing *how* the system components interact and are structured.**

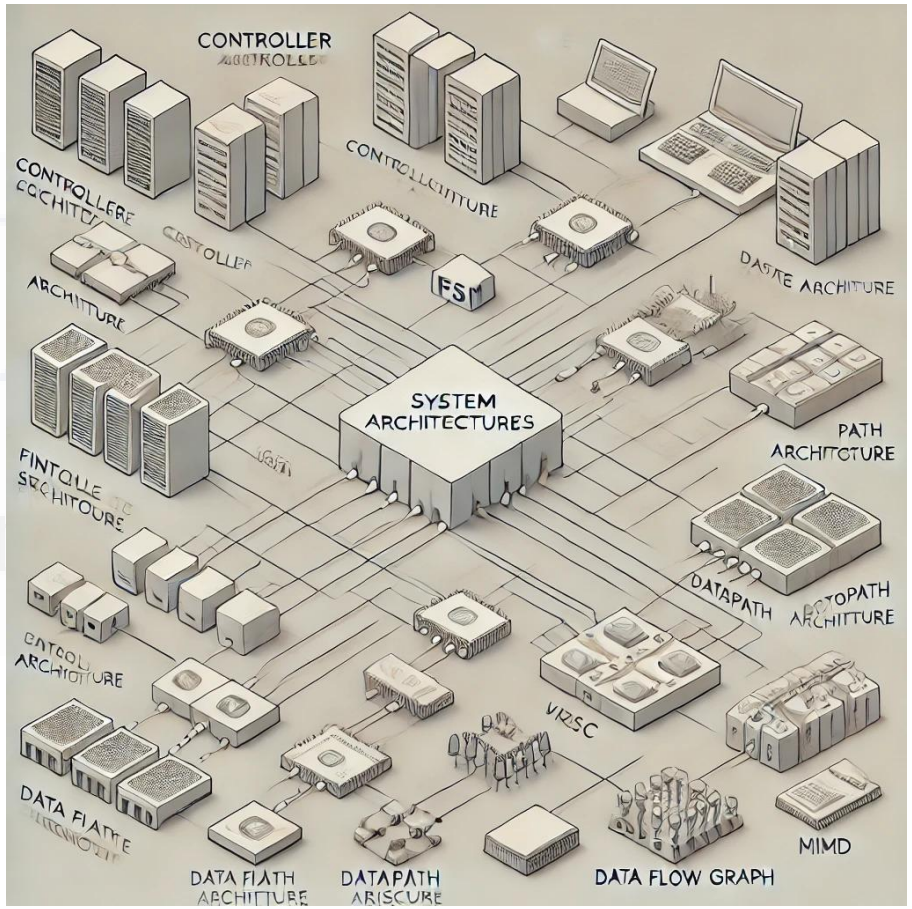
•**Phase 3: Transitioning Models**

As design evolves, switching between models is necessary, but the complexity increases.

3.1 Hardware Software Co-Design and Program Modelling

2. Selecting the Architecture:

1. **Controller Architecture:** Finite State Machine (FSM).
2. **Datapath Architecture:** Data Flow Graph (DFG).
3. **FSMD:** Combination of controller + datapath.
4. **Other Architectures:** CISC, RISC, VLIW, SIMD, MIMD.



1.Controller Architecture (FSM):

Uses states and transitions to control system behavior, ideal for managing sequential tasks.

2.Datapath Architecture (DFG):

Represents computations through data flow, where nodes are operations, and edges are data dependencies.

3.FSMD:

Combines FSM and Datapath, managing control and data flow together.

4.CISC:

Complex instructions that perform multiple operations in one, reducing memory use.

5.RISC:

Simpler instructions, requiring more operations but improving execution speed.

6.VLIW: Very Long Instruction Word:

Packs multiple operations into one instruction for parallel execution.

7.SIMD: Single Instruction, Multiple Data

Applies one instruction to multiple data elements simultaneously, useful for parallel tasks.

8.MIMD: Multiple Instruction, Multiple Data:

Multiple processors executing different instructions on different data at the same time.

Let me know if you'd like more details on any of these!

3. Selecting the Language:

- **Software Languages:** C, C++, Java.
- **Hardware Languages:** VHDL, Verilog.

Language	Type	Best For	Use Case
C	Software	Performance and low-level control	Real-time systems, device drivers, microcontrollers
C++	Software	Object-oriented design for complex systems	Complex embedded systems, modular design
Java	Software	High-level embedded systems	IoT devices, smart appliances, cloud-based systems
VHDL	Hardware	Hardware description and design	FPGA, ASIC design, custom hardware creation
Verilog	Hardware	Hardware description and design	FPGA, ASIC design, custom hardware creation

4. Partitioning System Requirements:

- Hardware vs Software trade-offs.

- **Hardware** is best for tasks requiring **high performance** and **low power**, but it comes with a **higher cost** and **longer development time**.
- **Software** is more **flexible**, easier to modify, and quicker to develop, but may sacrifice performance and power efficiency.

Example

- **Hardware**: Used for time-critical, performance-sensitive tasks like **temperature measurement**.
- **Software**: Used for flexible tasks like **user control** and **scheduling**, which don't require real-time processing.

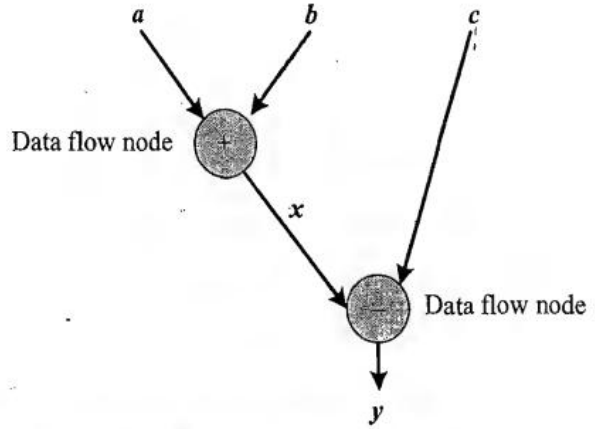
3.2 Computational Models in Embedded System Design

3.2 Computational Models in Embedded System Design

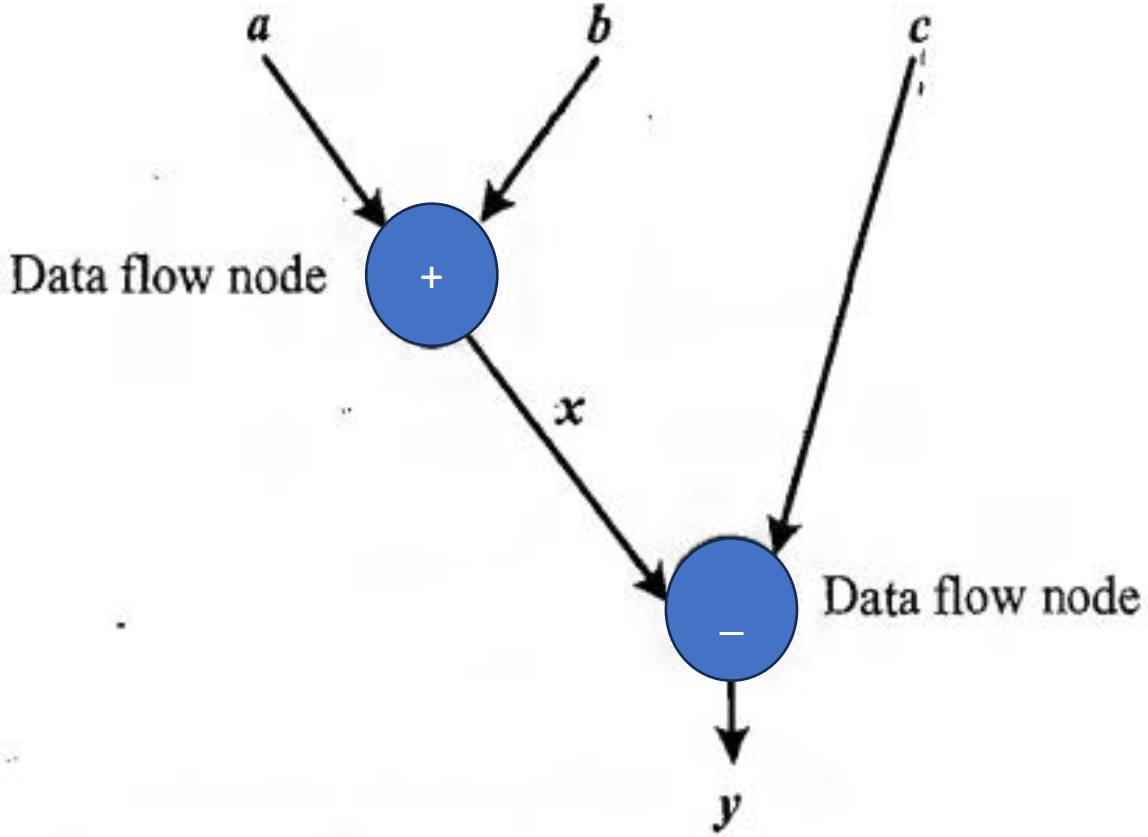
The **Data Flow Graph (DFG)** model is commonly used to represent computations in embedded systems, especially for tasks like Digital Signal Processing (DSP) and computational operations **where data is transformed through various functional blocks.**

1.Data Flow Graph (DFG) Model:

1. Data-driven model where operations transform input to output.
2. **Applications:** DSP and computational tasks.
3. **Example:** Acyclic DFG for mathematical operations.



- **Operations** are represented by nodes in the graph, and
- **Data flow** is represented by directed edges/arrows between nodes.



Example: Consider a simple mathematical operation as an example:

Example of an Acyclic Data Flow Graph (DFG):

Let's say we have a system that needs to compute the sum of squares of two numbers a and b.

Mathematical operation: $y = (a^2) + (b^2)$

1. **Node 1** (representing the square of `a`):
   - **Input:** `a`
   - **Output:** a^2

2. **Node 2** (representing the square of `b`):
   - **Input:** `b`
   - **Output:** b^2

3. **Node 3** (representing the addition of squares):
   - **Input:** a^2 and b^2
   - **Output:** $y = a^2 + b^2$

Example: Audio Signal Processing System

In an embedded system, you might have a **microcontroller** that processes incoming audio signals. Let's say you want to apply a **filter** to the audio signal, **square the values** of the signal to increase amplitude, and then **sum** the results for further processing.

Scenario:

1.Input: Raw audio signal sampled at certain intervals.

2.Goal: Square the signal values and then sum them.

Why Data Flow Graph (DFG)?

- The **DFG** helps visualize how data (audio signals) flows through various processing nodes.
- It allows you to represent **independent operations** that can run in parallel, such as squaring and summing the signals.

Steps in the Data Flow Graph (DFG):

- **Node 1**: Take the raw audio signal input (let's call it `a`) and apply a **squaring operation**. The output of this node is a^2.

Node 2: Apply the **same operation** to another audio signal or the same one (e.g., `b`), producing b^2.

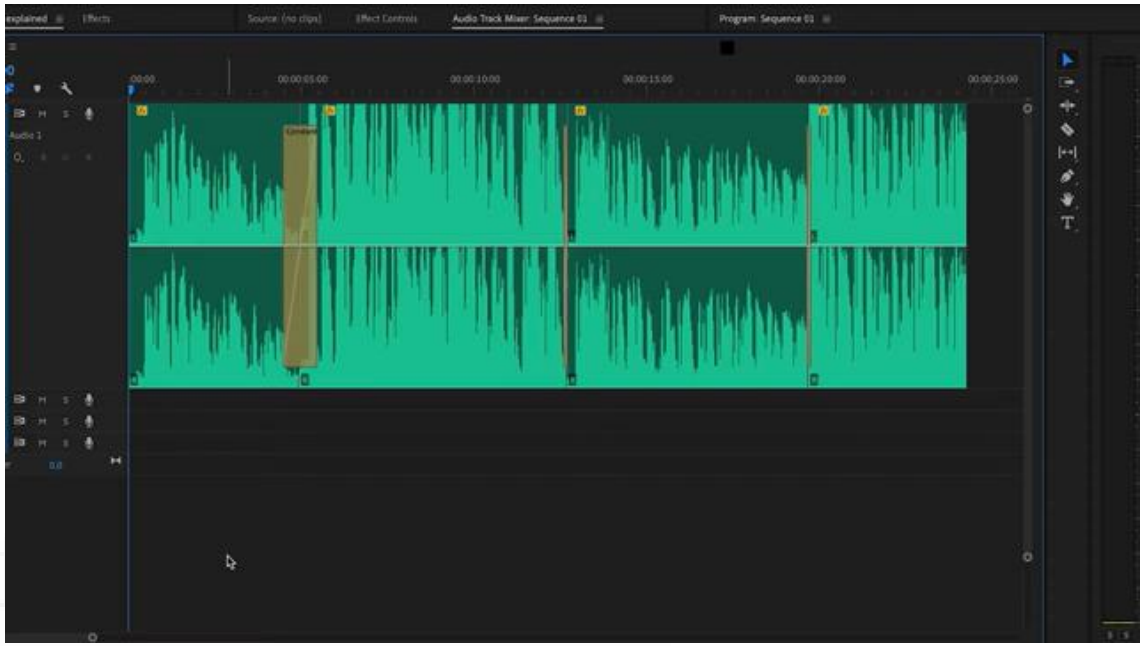
Node 3: Add the squared values $a^2 + b^2$ together to produce a final result, which can be used for further signal processing.

Note:

What You Will Hear:

•**Normal Audio**: A clear, regular sound (such as a sine wave or speech).

•**Squared Audio**: The audio will sound more distorted with louder peaks because squaring the signal intensifies the higher parts of the waveform. This could make the sound appear "compressed" or have a more clipped, amplified nature.



2. Control Data Flow Graph (CDFG) Model

The **Control Data Flow Graph (CDFG)** extends the basic Data Flow Graph (DFG) by incorporating **control operations** or **decision-making** processes into the data flow. This type of graph is used when the **flow of data depends on conditions or decisions**, such as **if-else** conditions, loops, or branches in the code.

1. Extends **DFG model** by incorporating **control operations (conditionals)**.
2. Uses **decision nodes (diamonds)** to represent conditional execution.
3. Example: **Image processing in a digital camera**, where the user selects **JPEG, TIFF, BMP** format.

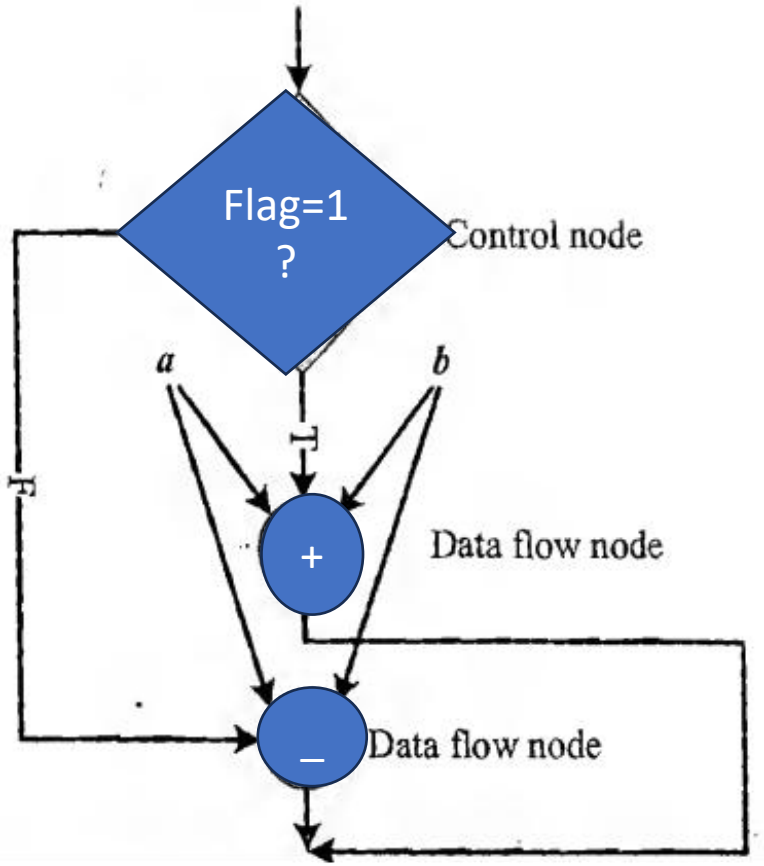
2. Control Data Flow Graph (CDFG) Model

What is a CDFG?

In a **CDFG**, you have both:

1.Data flow (operations on data, like addition, subtraction, etc.)

2.Control flow (decision-making, like conditions and branches).



If flag = 1, x = a + b; else y = a - b;

Practical Example: Image Processing in a Digital Camera

In the context of **image processing**, let's say a digital camera has an option for the user to select the image format (JPEG, TIFF, or BMP).

•The camera must **decide** which image format to save based on the user's choice. This is a **control operation** (decision-making).

•The **data flow** consists of processing the image data (e.g., resizing, compression), and the **control flow** determines whether the image should be saved as **JPEG, TIFF, or BMP.**

Example Process:

1.User Input (control node):

1. The user selects the format (JPEG, TIFF, BMP).

2.Condition (decision node):

1. If the user selects **JPEG**, compress the image using the JPEG algorithm.
2. If the user selects **TIFF**, save the image with no compression.
3. If the user selects **BMP**, save the image in uncompressed form.

3.Data Operations (data flow nodes):

1. The image data is processed based on the selected format.

3. State Machine Model (FSM):

A **Finite State Machine (FSM)** is used to model **reactive systems**, which can be in one state at a time and **transition between states** based on events or inputs.

The FSM is defined by a set of **states**, **events**, and **actions**.

1. Represents reactive systems through states, events, and actions.
2. **Example:** Seat Belt Warning System.

- **States**: Represent the different conditions or configurations of the system.
- **Events**: Conditions that cause the system to transition from one state to another.
- **Actions**: Operations that are performed when transitioning between states or during certain events.

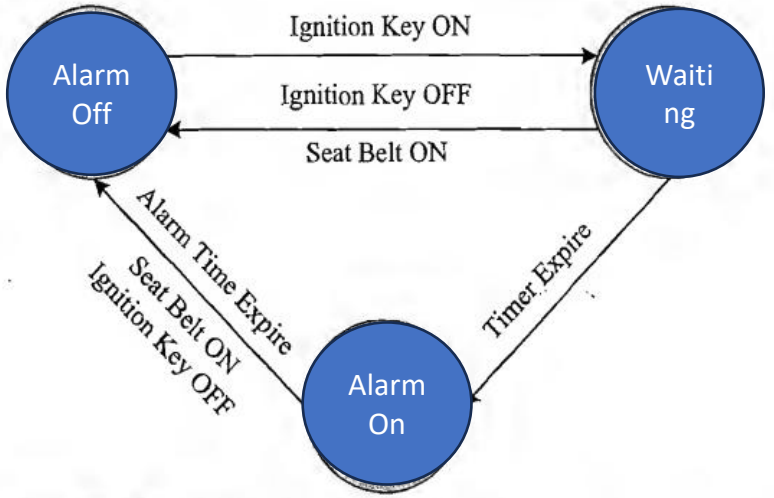


Fig. 7.3 FSM Model for Automatic seat belt warning system

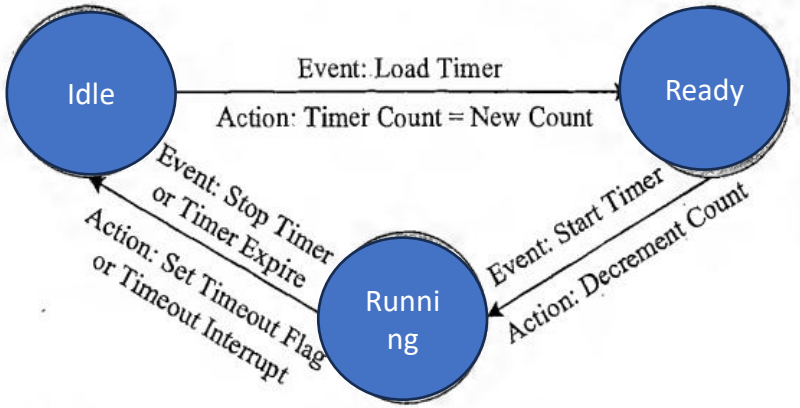
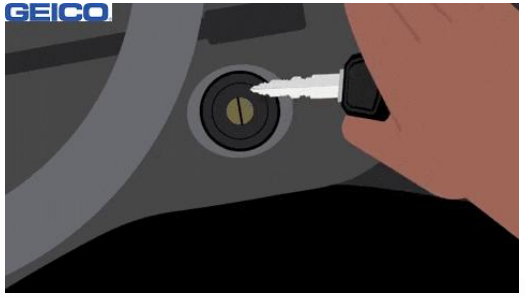


Fig. 7.4 FSM Model for timer

Example: Seat Belt Warning System



States: Alarm Off: When the ignition key is on, and the seatbelt is either on or off, the alarm may or may not be triggered.

Alarm On: If the ignition is on, but the seatbelt is not fastened, the alarm goes off. Waiting: The **system waits for the seatbelt to be fastened** or the **ignition key to be turned off**.

Events:
Ignition Key ON/OFF: This triggers a transition between the Waiting and Alarm states.

Seat Belt ON/OFF: This can cause the system to transition from Alarm On to Alarm Off.

Actions:
- If the seatbelt is **not fastened,** the action would be to turn on the alarm.
- If the seatbelt is **fastened**, the action would be to turn off the alarm.

4. Sequential Program Model:

1. Executes tasks in a defined sequence.
2. Tools: FSMs, Flow Charts.

A sequential model refers to an approach where tasks are executed one after another, **following a predefined order.**

This is one of the simplest forms of programming logic, where each operation is completed **before moving on to the next step.**

SEQUENTIAL PROGRAM MODEL FOR SEAT BELT WARNING SYSTEM

```
# define ON 1
# define OFF 0
# define YES 1
# define NO 0
Void seat_belt_warn( )
{
wait_10sec( );
if( check_ignition_key(  ) = = ON)
{
if( check_ignition_key(  ) = = OFF)
{
set_timer(5);
Start_alarm( );
While ((check_seat_belt ( ) = = OFF) && ( check_iginition_key ( ) = = OFF) && (timer_expire ( ) =
    =NO));
Stop_alarm ( );
}
}
}
```

This program defines a seat belt warning system that:

- **Waits for 10 seconds.**
- **Checks if the ignition key is ON.**
- **If the ignition key is then turned OFF, a 5-second timer is set, and an alarm is started.**
- **The alarm continues to sound as long as the seat belt is not fastened, the ignition key remains OFF, and the timer has not expired.**
- **Once the conditions are no longer met, the alarm stops.**

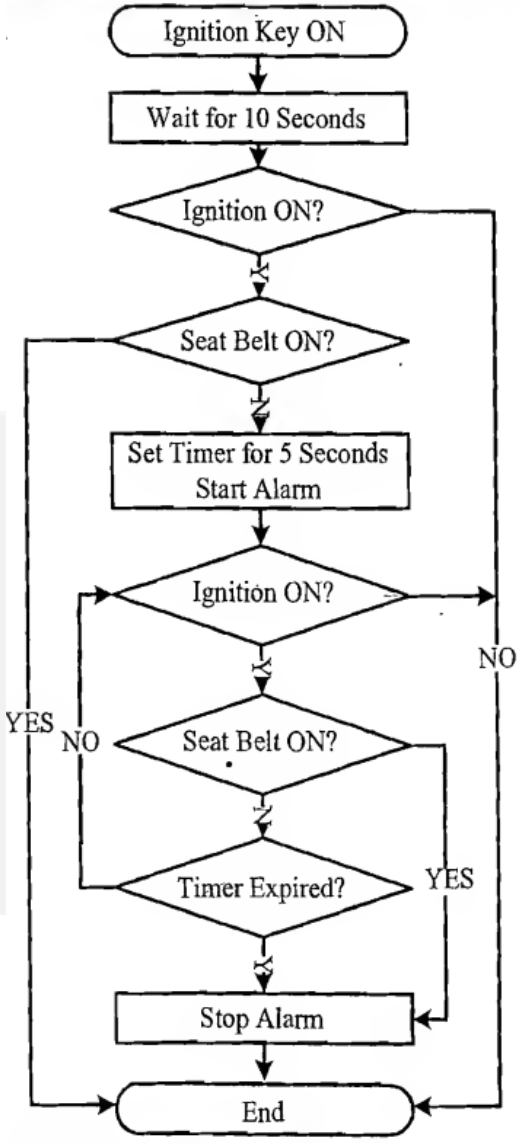


Fig. 7.7 **Sequential Program Model for seat belt warning system**

Example-1

Question

Design an Automatic Tea/Coffee Vending Machine using FSM Model

A **tea/coffee vending machine** operates based on a **Finite State Machine (FSM) model** with the following requirements:

1. The machine starts in **'Wait for Coin'** state.
2. The user **inserts a ₹5 coin** to initiate the vending process.
3. After inserting the coin, the system transitions to **'Wait for User Input'** state.
4. The user can select:
   - **'Tea'** → Transitions to **'Dispense Tea'** state.
   - **'Coffee'** → Transitions to **'Dispense Coffee'** state.

- **Cancel’** → Coin is returned, and state transitions back to **‘Wait for Coin’**.

1. Once the drink is dispensed, the system resets back to **‘Wait for Coin’**.
2. Additional conditions may include:
   - **Timeout in ‘Wait for User Input’ state** (if no input, coin is returned).
   - **Error Handling** for **‘Water Not Available’** or **‘Mix Not Available’**.

Solution

FSM Model for Automatic Tea/Coffee Vending Machine

States:

1. **Wait for Coin**: The machine is idle, waiting for the user to insert a coin (₹5).
2. **Wait for User Input**: After the coin is inserted, the machine waits for the user to make a choice.
3. **Dispense Tea**: If the user selects "Tea", the machine dispenses tea.
4. **Dispense Coffee**: If the user selects "Coffee", the machine dispenses coffee.
5. **Return Coin**: If the user selects "Cancel", the coin is returned, and the machine goes back to the "Wait for Coin" state.
6. **Timeout**: If no input is received in a defined time while in the "Wait for User Input" state, the machine returns the coin.
7. **Error Handling**: In the case of a shortage of ingredients like water or mix, the system transitions to an error state and informs the user.

FSM Process Explanation:

1.Start (Wait for Coin): The machine begins in this state.

1. The machine is idle and waits for the user to insert a ₹5 coin.

2.Transition to Wait for User Input: Once the coin is inserted, the system transitions to the "Wait for User Input" state.

3.User Input Options: The user can select one of the following options:

1. **Tea**: The system transitions to the "Dispense Tea" state, and tea is dispensed. After dispensing, it goes back to "Wait for Coin".
2. **Coffee**: The system transitions to the "Dispense Coffee" state, and coffee is dispensed. After dispensing, it goes back to "Wait for Coin".
3. **Cancel**: If the user presses "Cancel", the coin is returned, and the machine goes back to the "Wait for Coin" state.

4.Timeout in Wait for User Input: If the user does not make a selection in a given time (e.g., 30 seconds), the coin is returned automatically, and the machine resets to the "Wait for Coin" state.

5.Error Handling: If there is an issue with ingredients, such as no water or mix, the system handles the error by transitioning to an error state and informing the user that the ingredients are unavailable.

Flowchart Representation:

1.Wait for Coin

1. Coin Inserted → **Wait for User Input**

2.Wait for User Input

1. Tea Selected → **Dispense Tea**
2. Coffee Selected → **Dispense Coffee**
3. Cancel Selected → **Return Coin** → **Wait for Coin**
4. Timeout → **Return Coin** → **Wait for Coin**

3.Dispense Tea / Dispense Coffee

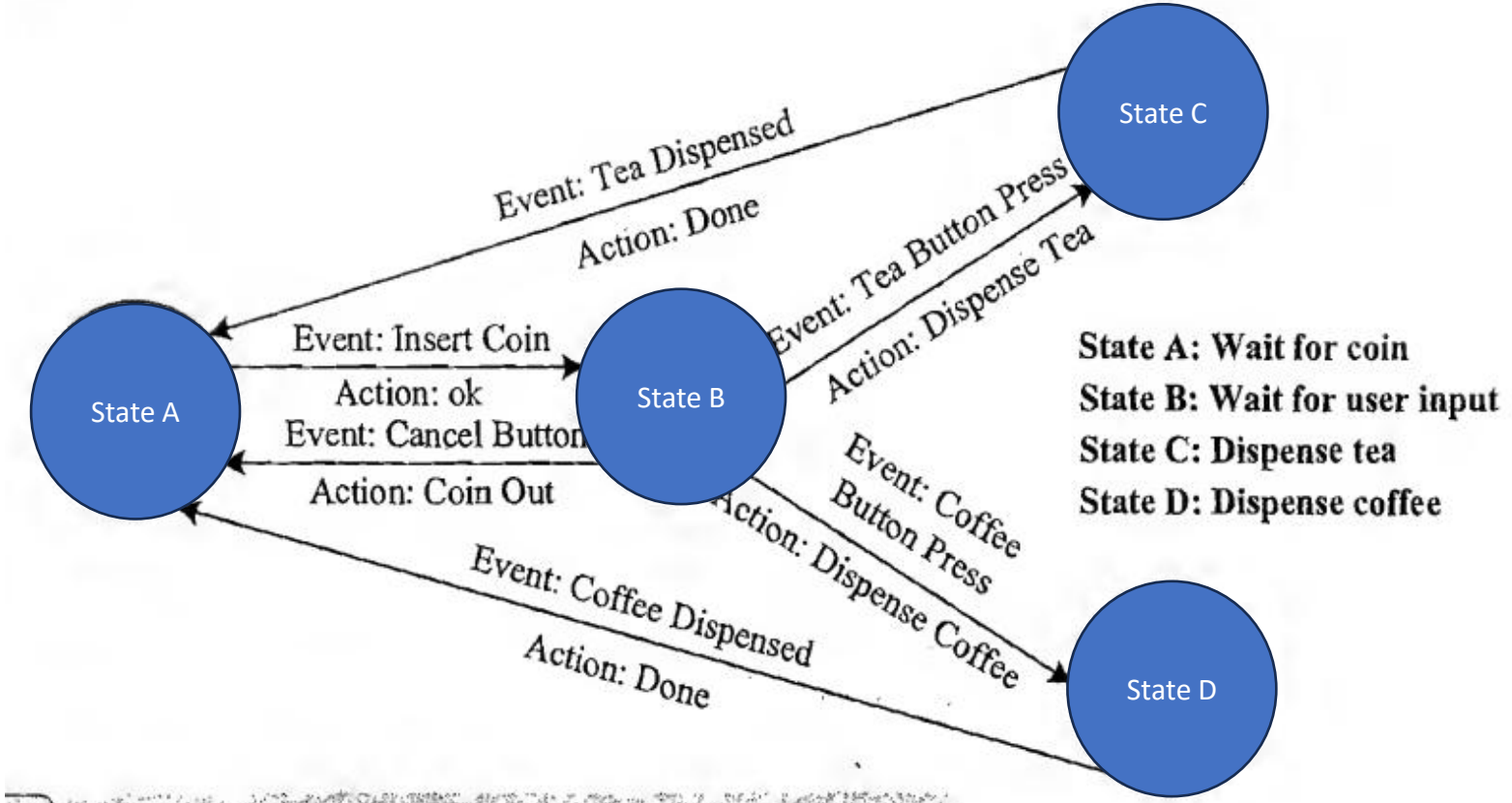
1. Dispense Complete → **Wait for Coin**

4.Error Handling (Water/Mix Unavailable)

1. Error Detected → **Error State** → **Inform User** → **Wait for Coin**

FSM State Transitions:

Current State	Event (Input)	Next State	Action
Wait for Coin	Insert ₹5 coin	Wait for User Input	Accept Coin
Wait for User Input	Press ‘Tea’	Dispense Tea	Start Tea Dispensing
Wait for User Input	Press ‘Coffee’	Dispense Coffee	Start Coffee Dispensing
Wait for User Input	Press ‘Cancel’	Wait for Coin	Return Coin
Dispense Tea	Tea Dispensed	Wait for Coin	Reset System
Dispense Coffee	Coffee Dispensed	Wait for Coin	Reset System
Wait for User Input	Timeout (No Selection)	Wait for Coin	Return Coin
Any State	Error (No Water/Mix)	Error State	Display Error
Error State	Problem Resolved	Wait for Coin	Reset System



7.5 FSM Model for Automatic Tea\Coffee Vending Machine

Concurrent/Communicating Process Model

1.Definition:

1. Models concurrently executing tasks/processes instead of sequential execution.
2. More efficient for tasks involving I/O waiting or sleep states.

2.Advantages:

1. Better CPU utilization by switching tasks during wait or sleep modes.
2. Allows handling multiple subtasks simultaneously.

3.Challenges:

Requires additional overheads for task scheduling, synchronization, and communication.

•**Example: Seat Belt Warning System**

•**Tasks divided into five concurrent processes:**

1. **Wait Timer Task** – Waits for 10 seconds.
2. **Ignition Key Status Task** – Monitors ignition key state.
3. **Seat Belt Status Task** – Monitors seat belt status.
4. **Alarm Control Task** – Starts and stops the alarm.
5. **Alarm Timer Task** – Waits for 5 seconds before stopping

1. Synchronization Using Events

2. Wait Timer Expiry Event: Set when the wait timer expires.

3. Ignition Key Events:

- •ignition_on (Set when ignition is ON, reset when OFF).
- •ignition_off (Set when ignition is OFF, reset when ON).

4. Seat Belt Events:

- •seat_belt_on (Set when seat belt is ON, reset when OFF).
- •seat_belt_off (Set when seat belt is OFF, reset when ON).

5. Alarm Timer Events:

- •alarm_timer_start (Set when the alarm is started).
- •alarm_timer_expire (Set when alarm timer expires).

6. Alarm Activation Logic

- Alarm starts **only if**:
  - Wait timer expires.
  - Ignition is ON.
  - Seat belt is OFF.
- Alarm stops **when any of the following occur**:
  - Alarm timer expires.
  - Ignition turns OFF.
  - Seat belt is fastened.

7. Real-Time System Implementation

- Concurrent processing models are widely used in **real-time systems.**
- Common techniques for process communication:
  - **Shared Memory**
  - **Message Passing**
  - **Events** (used in the above example).

8. Flexibility in Implementation

- The example illustrates one approach; other methods can be used.

Example: Traffic Light Control System Using Concurrent Processing

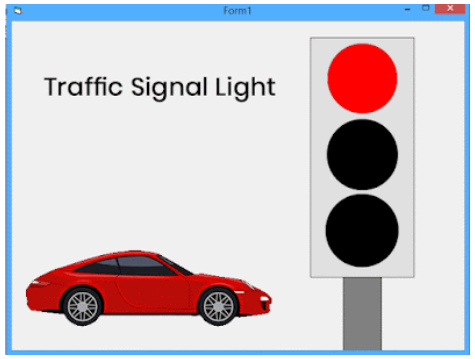
Scenario

A traffic light system at an intersection has three signals: **Red, Yellow, and Green**.

Instead of running sequentially, we use a **concurrent processing model** to manage them efficiently.

Tasks in the Concurrent Model

1. **Red Light Task** – Keeps the red light ON for a fixed duration.
2. **Green Light Task** – Keeps the green light ON for a fixed duration.
3. **Yellow Light Task** – Turns on the yellow light before switching to red.
4. **Timer Task** – Manages timing for each light.
5. **Pedestrian Signal Task** – Synchronizes with the red light for pedestrian crossing.



Synchronization Using Events

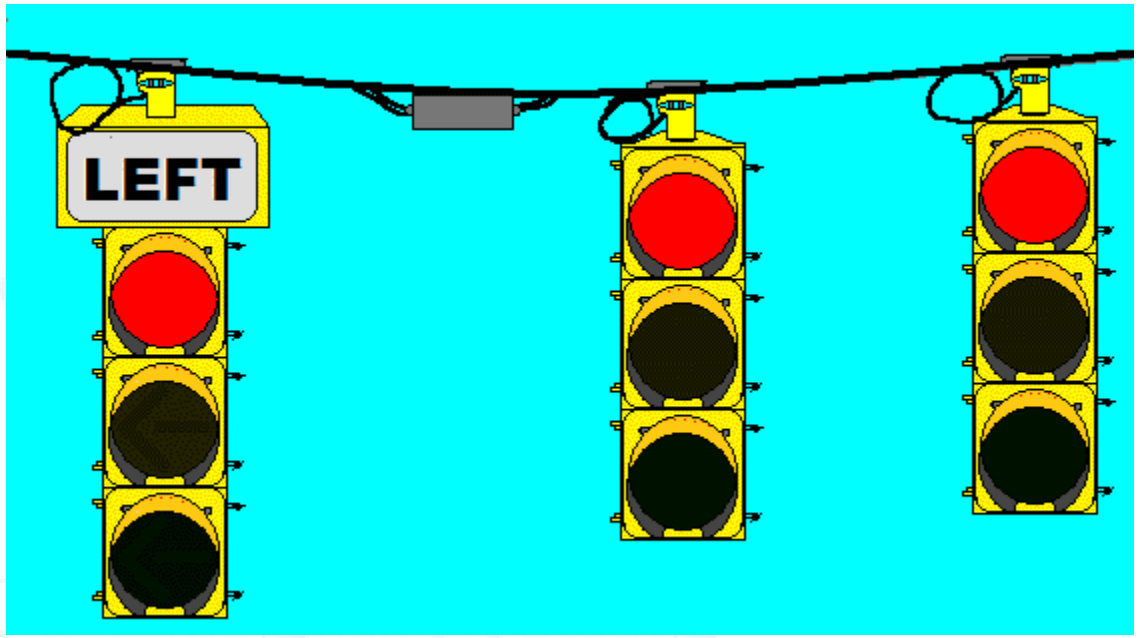
- red_light_on → Set when the red light is active.
- green_light_on → Set when the green light is active.
- yellow_light_on → Set when the yellow light is active.
- timer_expire → Set when a light duration is complete.
- pedestrian_crossing → Set when pedestrians can cross.

How It Works

1.Red Light Turns ON → red_light_on event is set, and pedestrians can cross.
2.Timer Expires → timer_expire triggers the green light.
3.Green Light Turns ON → green_light_on event is set, allowing vehicles to move.
4.Timer Expires → timer_expire triggers the yellow light.
5.Yellow Light Turns ON → yellow_light_on event is set (brief transition phase).
6.Timer Expires → timer_expire triggers the red light again.
7.Cycle Repeats

Advantages of Concurrent Processing

✅ No idle waiting – tasks run in parallel.
✅ Efficient synchronization between lights and pedestrian signals.
✅ Handles real-time scenarios effectively.



Object-Oriented Model

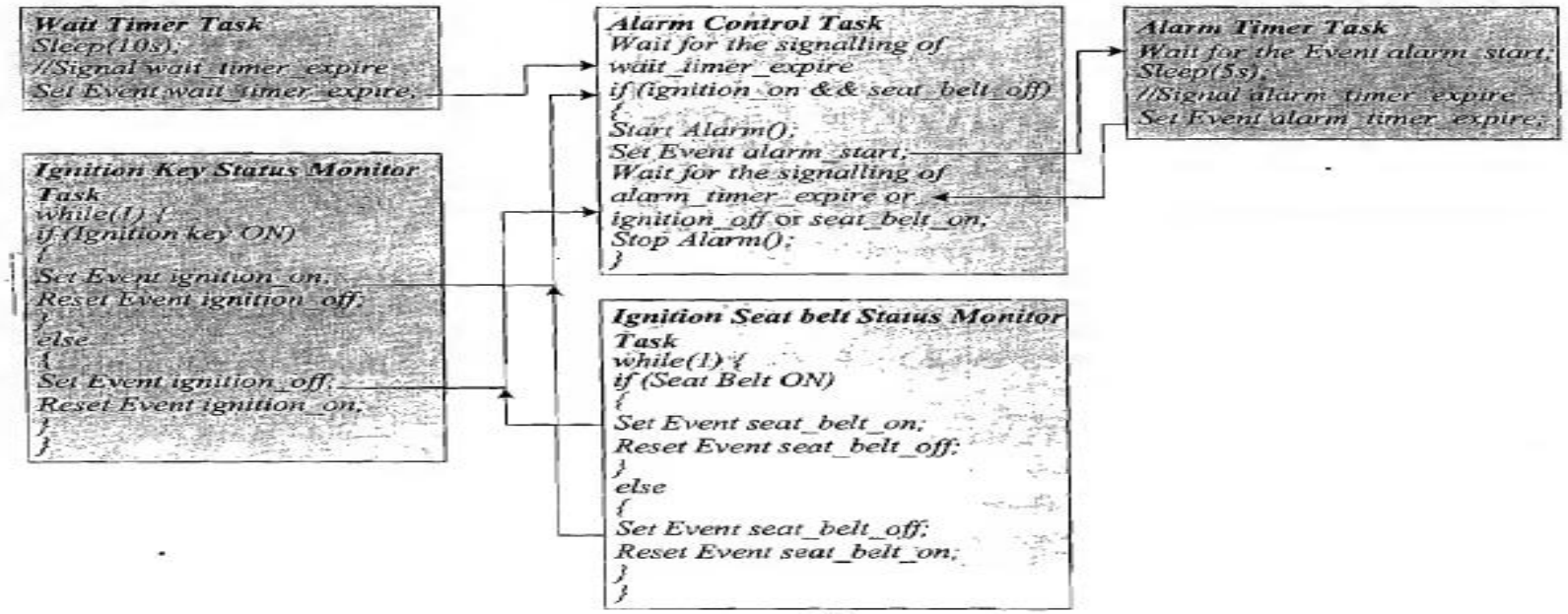
The **object-oriented model** is a system design approach that breaks down complex software requirements into **objects**. Each **object** represents a specific part of the system with unique **behavior** and **state**.

Key Concepts

- **Class** → A blueprint that defines the state (variables) and behavior (functions) of objects.
- **Encapsulation** → Hides internal details, exposing only necessary functionalities.
- **Access Modifiers**:
  - **Private** → Accessible only within the class.
  - **Public** → Accessible inside and outside the class.
  - **Protected** → Accessible within the class and its derived classes.
- **Inheritance** → A class can derive properties and methods from another class.

```
Create and initialize events
wait_timer_expire, ignition_on, ignition_off,
seat_belt_on, seat_belt_off,
alarm_timer_start, alarm_timer_expire
Create task Wait Timer
Create task Ignition Key Status Monitor
Create task Seat Belt Status Monitor
Create task Alarm Control
Create task Alarm Timer
```

(a)



(b)

Fig. 7.8 **(a) Tasks for 'Seat Belt Warning System' (b) Concurrent processing Program model for 'Seat Belt Warning System'**

Tasks for 'Seat Belt Warning System'

Create and initialize events

- wait_timer_expire, ignition_on, ignition_off
- seat_belt_on, seat_belt_off
- alarm_timer_start, alarm_timer_expire

Create tasks

1.Wait Timer Task

- Sleeps for 10 seconds
- Signals wait_timer_expire event

2.Ignition Key Status Monitor Task

- Runs in a loop
- If **ignition key is ON**
  - Sets ignition_on event, resets ignition_off event
- Else
  - Sets ignition_off event, resets ignition_on event

- **Seat Belt Status Monitor Task**
- Runs in a loop
- If **seat belt is ON**
  - Sets seat_belt_on event, resets seat_belt_off event
- Else
  - Sets seat_belt_off event, resets seat_belt_on event

- **Alarm Control Task**
- Waits for wait_timer_expire event

- If **ignition is ON and seat belt is OFF**
  - Starts alarm
  - Sets alarm_timer_start event
- Waits for alarm_timer_expire, ignition_off, or seat_belt_on event
- Stops alarm

Alarm Timer Task

- Waits for alarm_timer_start event
- Sleeps for 5 seconds
- Signals alarm_timer_expire event

Object-Oriented Model for Seat Belt Warning System

Instead of defining independent tasks and events, we define **objects and classes** to encapsulate system behavior.

Key Objects & Classes

1.Class: Timer

- Attributes: duration, expired
- Methods: start_timer(), check_expiry(), reset_timer()

2.Class: IgnitionKey

- Attributes: status (ON/OFF)
- Methods: turn_on(), turn_off(), get_status()

3.Class: SeatBelt

- Attributes: status (ON/OFF)
- Methods: fasten(), unfasten(), get_status()

- **Class: Alarm**
- Attributes: status (ON/OFF)
- Methods: activate(), deactivate(), is_active()
- **Class: SeatBeltWarningSystem (Main Controller)**
- Attributes:
  - ignition (IgnitionKey object)
  - seat_belt (SeatBelt object)
  - alarm (Alarm object)
  - wait_timer (Timer object)
  - alarm_timer (Timer object)
- **Methods:**
  - monitor_ignition() – Checks if ignition is ON
  - monitor_seatbelt() – Checks if seat belt is fastened
  - control_alarm() – Activates or deactivates alarm based on conditions

Advantages of Object-Oriented Model

- ✅ **Encapsulation**: Each class handles its own data and behavior.
- ✅ **Reusability**: Objects like Timer and Alarm can be reused in other systems.
- ✅ **Maintainability**: Code is modular, making updates easier.

CISC architecture uses complex instructions that perform multiple operations in a single step, reducing memory access but requiring additional silicon for decoding.

RISC architecture, in contrast, uses simple instructions that execute quickly and support extensive pipelining, relying on a large register file.

VLIW(Very Long Instruction Word) architecture employs multiple functional units, executing one instruction per unit in parallel. Parallel processing architectures, like SIMD and MIMD, enhance performance by executing multiple operations simultaneously.

SIMD: Single Instruction Multiple Data executes a single instruction across multiple data elements, while MIMD allows different instructions to run concurrently, forming the basis of multiprocessor systems.

•Which of the following programming model is best suited for modelling a data driven embedded system

(a) State Machine
(b) Data Flow Graph
(c) Harel Statechart Model
(d) None of these

•Which of the following programming model is best suited for modelling a Digital Signal Processing (DSP) embedded system

(a) Finite State Machine
(b) Data Flow Graph
(c) Object Oriented Model
(d) UML

•Which of the following architecture is best suited for implementing a Digital Signal Processing (DSP) embedded system

(a) Controller Architecture
(b) CISC
(c) Datapath Architecture
(d) None of these

•Which of the following is a multiprocessor architecture?

(a) SIMD
(b) MIMD
(c) VLIW
(d) All
(e) (a) and (b)
(f) (b) and (c)

•**Which of the following model is best suited for modelling a reactive embedded system?**

(a) Finite State Machine (FSM)
(b) DFG
(c) Control DFG
(d) Object Oriented Model

•**Which of the following models is best suited for modelling a reactive real time embedded system?**

(a) Finite State Machine
(b) DFG
(c) Control DFG
(d) Hierarchical/Concurrent Finite State Machine Model

•Which of the following model is best suited for modelling an embedded system demanding multitasking capabilities with data sharing?

(a) Finite State Machine
(b) DFG
(c) Control DFG
(d) Communicating Process Model

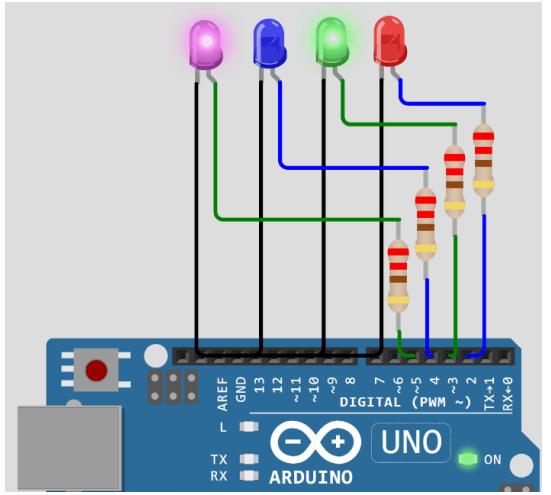
•Which of the following is a hardware description language?

(a) C
(b) System C
(c) VHDL
(d) C++
(e) (b) and (c)

3.3 Embedded Hardware Design and Development: Analog Components

1.Resistors: Current limiting.
2.Capacitors: Filtering, decoupling, RF matching.
3.Inductors: Ripple and noise filtering.
4.Diodes: Voltage clamping, rectification.
5.Transistors: Switching and amplification.

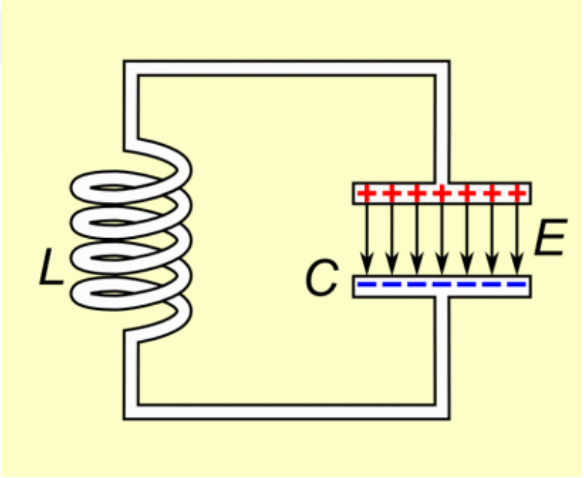
Analog electronic components like resistors, capacitors, diodes, inductors, OpAmps, and transistors are essential in embedded hardware design



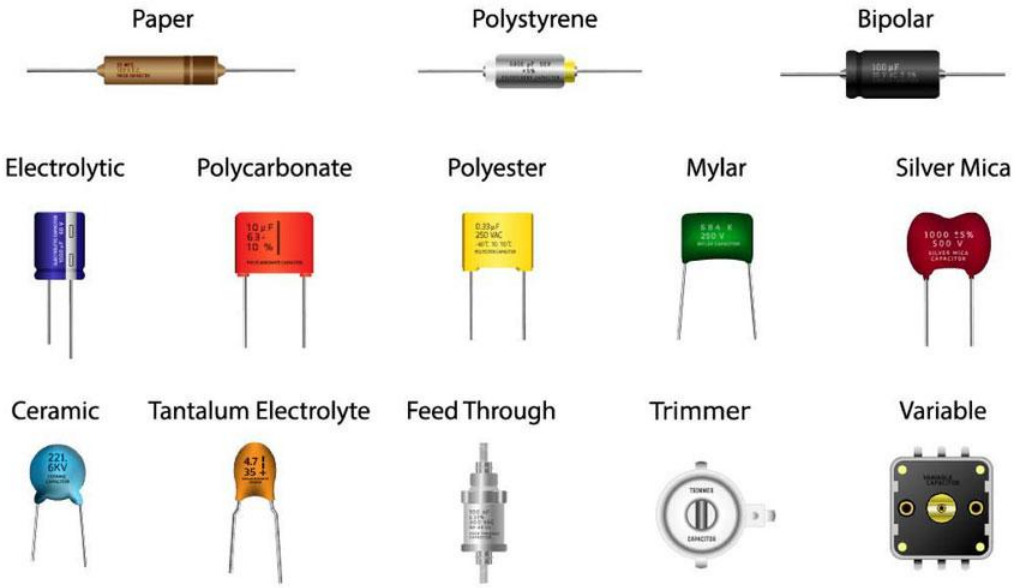
Resistors limit current flow, commonly used in interfacing LEDs and buzzers.

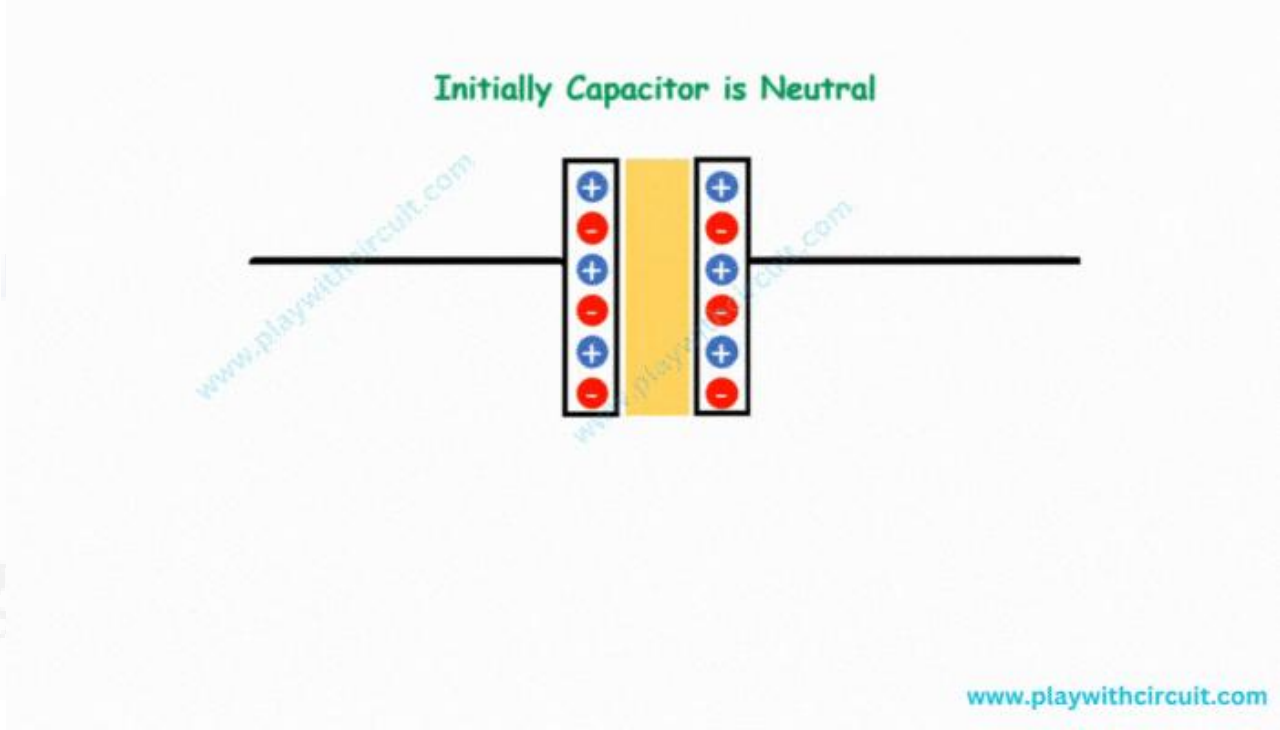
Capacitors, including electrolytic, ceramic, and tantalum types, are used in signal filtering, power decoupling, and reset circuits.

Inductors help filter power supply ripples and noise.



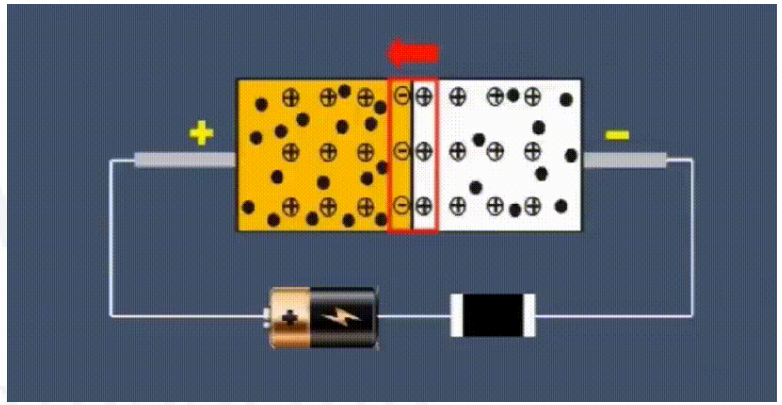
Capacitor Types



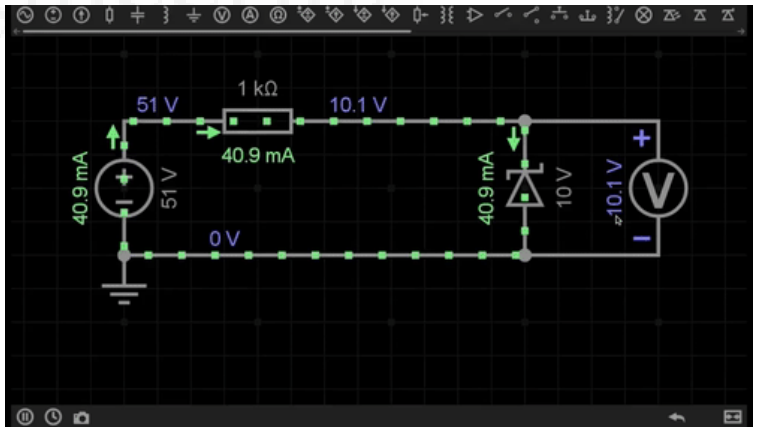


Diodes like P-N junction, Schottky, and Zener diodes serve in rectification, voltage clamping, and reverse polarity protection.

Schottky diodes have a lower voltage drop and faster switching speed than standard diodes.



Zener diodes regulate voltage by allowing reverse current beyond breakdown voltage.



3.4 Embedded Hardware Design and Development: Digital Components

1.Logic Gates: AND, OR, XOR, etc.

2.Buffers: Amplifies current or power.

3.Latches: Data storage.

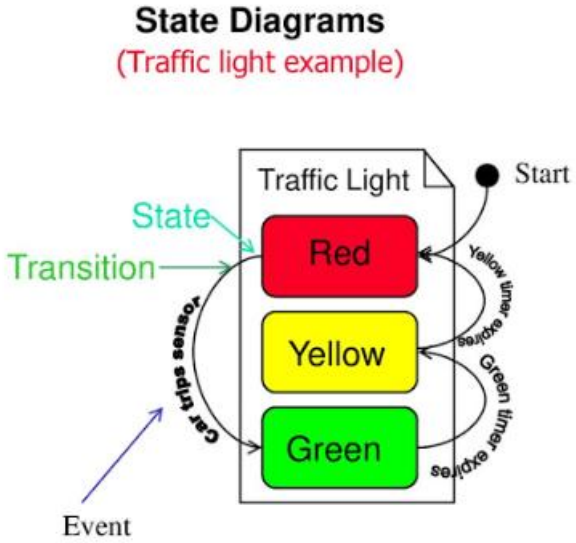
4.Decoders/Encoders: Address decoding and encoding.

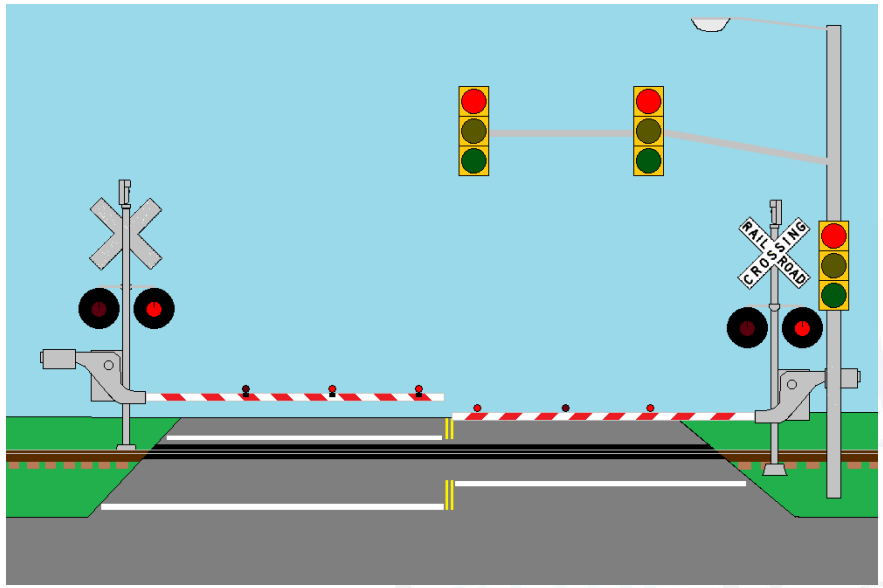
5.Multiplexers/Demultiplexers: Signal selection and distribution.

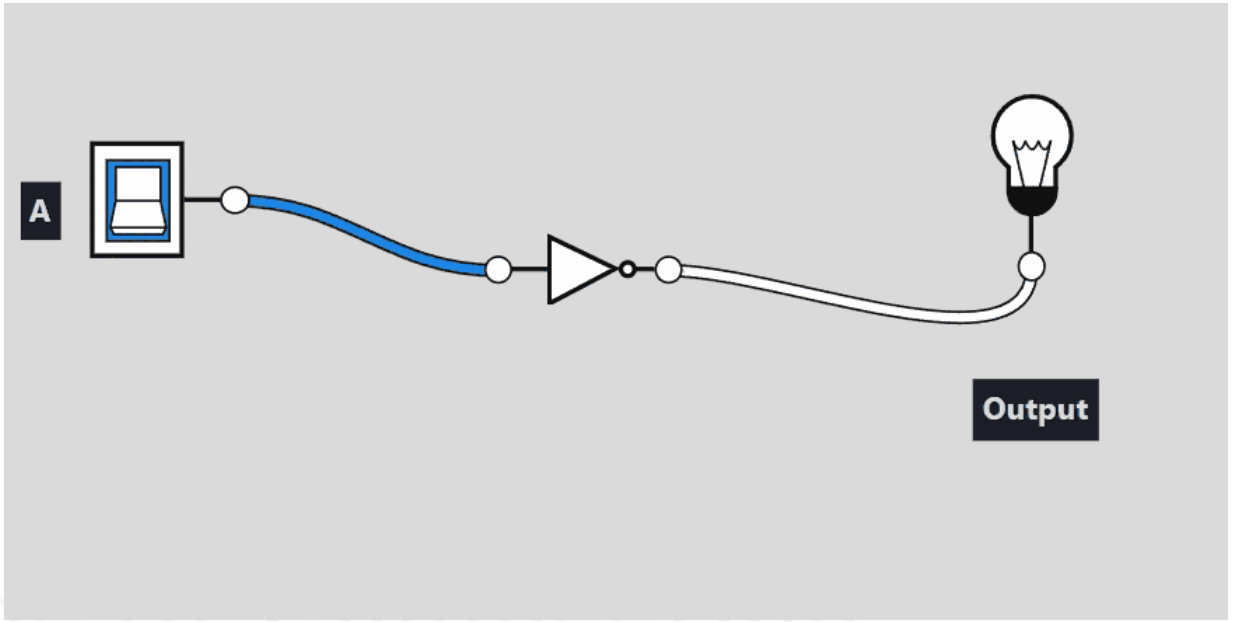
1. Logic Gates (AND, OR, XOR, etc.)

•**Purpose:** Perform basic logical operations in digital circuits.

•**Example:** An **AND gate** in a traffic light system that turns the green light on only if both the "sensor detected" and "no ongoing traffic" conditions are true.



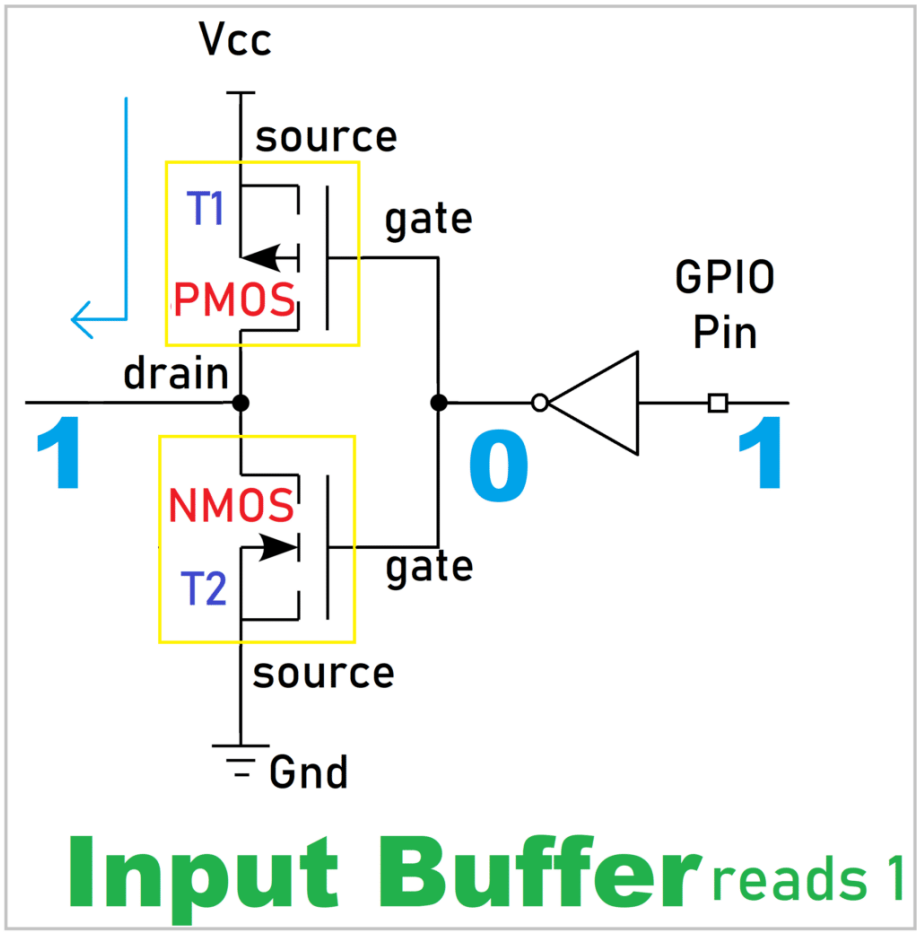

RAIL
CROSSING
ROAD

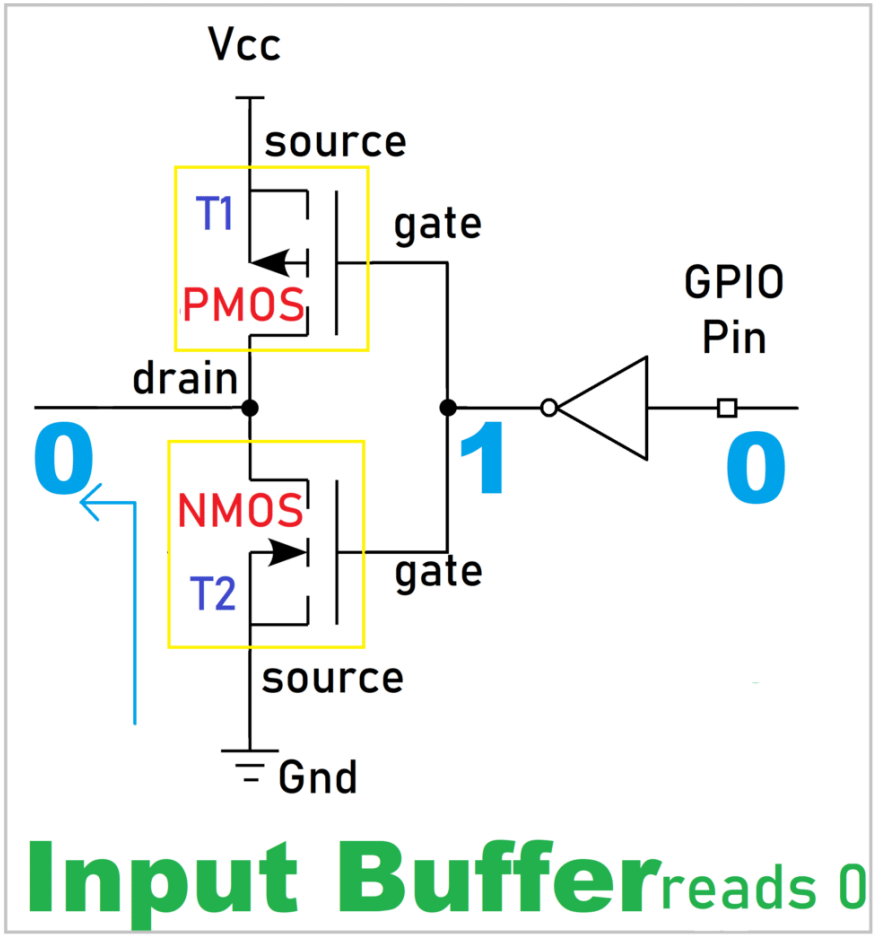

A
Output

2. Buffers

•**Purpose:** Amplify or maintain signal strength and current without altering the signal.

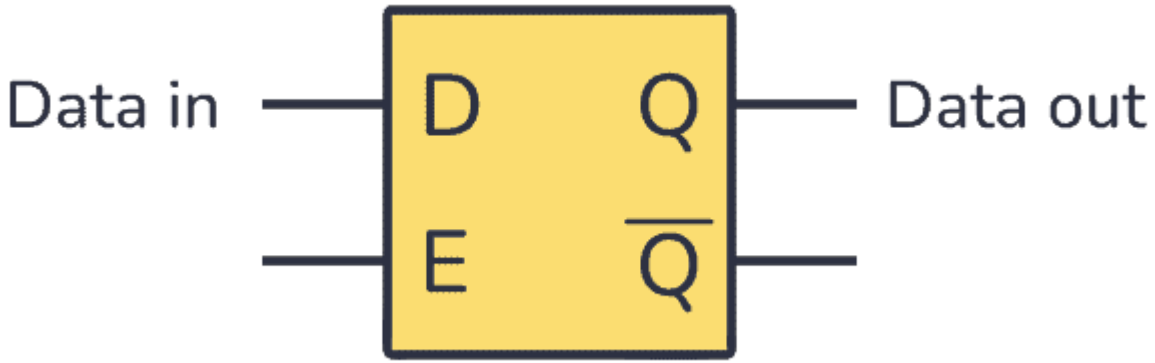
•**Example:** A buffer is used in a microcontroller to drive LEDs, ensuring the GPIO pins can handle the current required by the LEDs without damage.


Vcc
source
T1
PMOS
gate
GPIO
Pin
drain
1
0
1
NMOS
T2
gate
source
Gnd
Input Buffer reads 1


Vcc
source
T1
PMOS
gate
GPIO
Pin
drain
0
1
0
NMOS
T2
gate
source
Gnd
Input Buffer reads 0

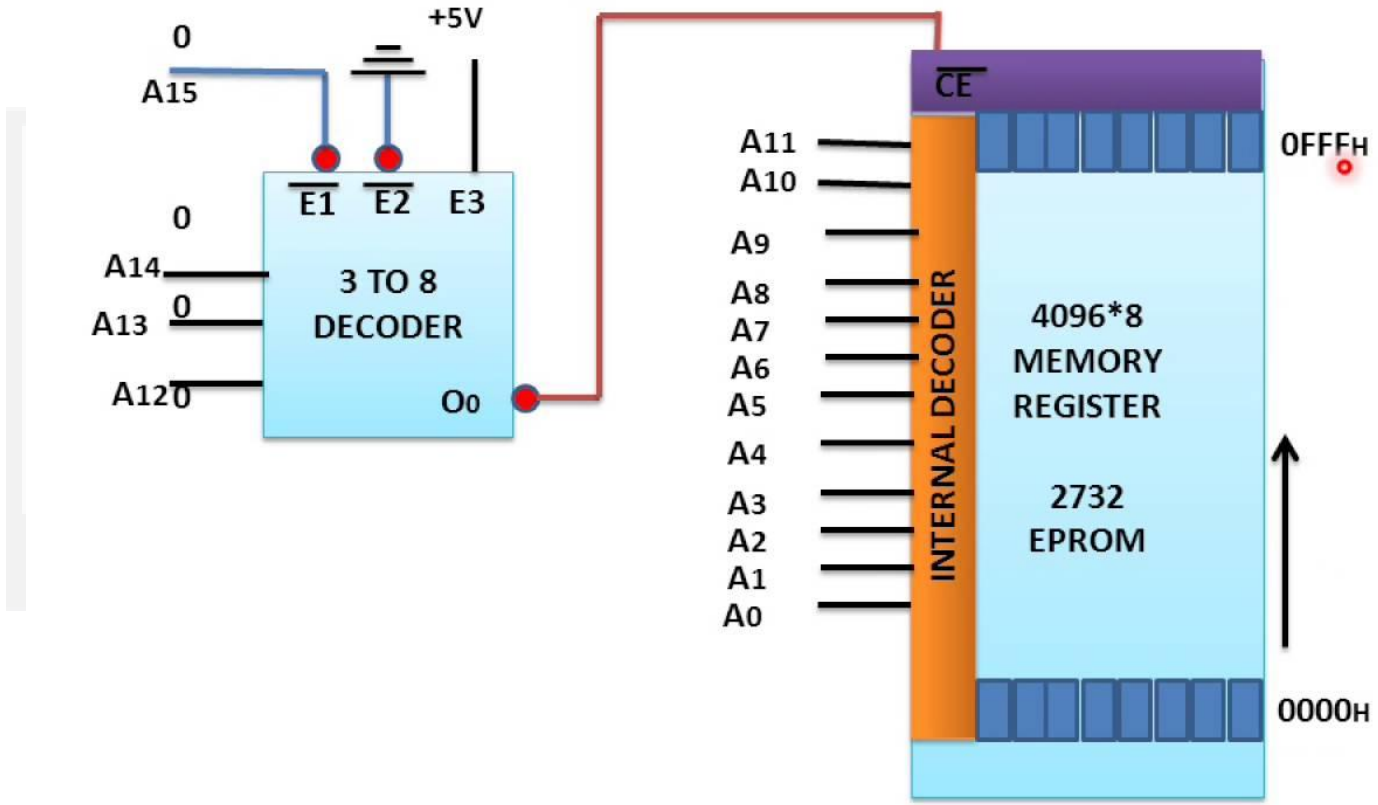
3. Latches

- **Purpose:** Store one bit of data and maintain its state until it is changed by a control signal.
- **Example:** A **D latch** stores data temporarily in a digital clock divider circuit.



4. Decoders/Encoders

- **Purpose:** Convert between different binary formats (e.g., address decoding or reducing input lines).
- **Example:** An **address decoder** in memory mapping that activates specific memory locations based on the input address.


+5V
0
A15
E1
E2
E3
0
A14
A13
0
A12
0
3 TO 8
DECODER
O0
CE
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
INTERNAL DECODER
4096*8
MEMORY
REGISTER
2732
EPROM
0FFFH
0000H

5. Multiplexers/Demultiplexers

- **Purpose:** Select or distribute signals to/from multiple sources.
- **Example:** A **multiplexer (MUX)** selects one of several sensor inputs to be sent to the microcontroller over a single data line, reducing the number of input pins required.

3.5 VLSI & Integrated Circuit Design

1.IC Types: SSI, MSI, LSI, VLSI.

2.Design Steps:

1.Specification, Design Entry, Simulation, Logic Synthesis.

2.Physical Layout, Timing Simulation.

3.VHDL: Describes behavior and structure of digital circuits.

VLSI (Very Large Scale Integration) is the process of integrating **thousands to millions of transistors** on a single chip. It allows the creation of complex digital circuits like microprocessors, memory chips, and SoCs (System on Chips).

IC Types: SSI, MSI, LSI, VLSI

1.SSI (Small Scale Integration):

1. **Description**: Refers to circuits that integrate a small number of logic gates (typically less than 10 gates) on a single chip.
2. **Example**: Simple logic gates like AND, OR, NOT gates, or a small flip-flop circuit.

Example of SSI:

A classic example of an SSI chip is the **7400 NAND gate IC**.

IC 7400 - Quad 2-Input NAND Gate

- **Type:** SSI (Small Scale Integration)
- **Package:** DIP-14 (Dual Inline Package)
- **Contains:** 4 independent 2-input NAND gates
- **Each gate uses ~4 transistors internally**
- **Total transistors:** Around 20–40

MSI (Medium Scale Integration):

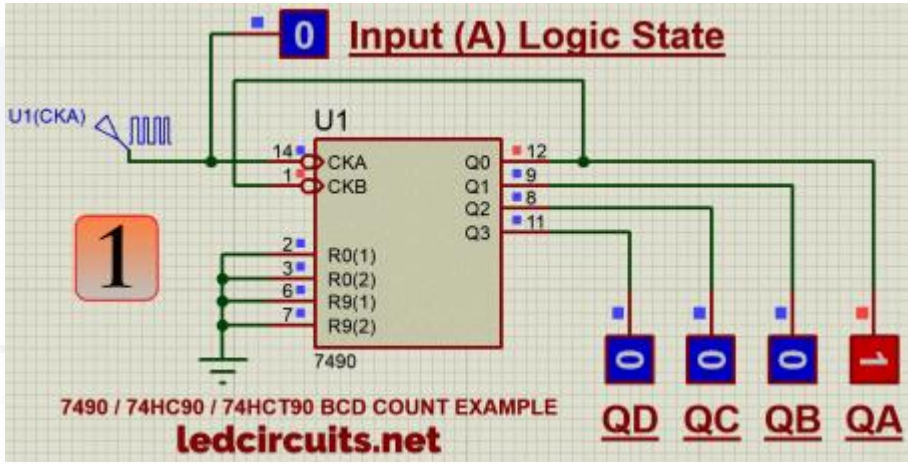
•**Description**: Refers to the integration of more logic gates, typically in the range of tens to hundreds of gates. MSI chips can perform more complex functions compared to SSI chips.

•**Example**: Adders, multiplexers, decoders, and small memory units.

MSI (Medium Scale Integration)

- **Transistors per chip:** 100 – 1,000
- **Logic gates per chip:** 10 to 100

- **Example:**
  - **IC 7490** – Decade Counter
  - **IC 74138** – 3-to-8 Line Decoder
  - **IC 74153** – Dual 4-to-1 Multiplexer



➤ These perform more complex logic operations than SSI.

LSI (Large Scale Integration):

•**Description**: LSI chips integrate thousands of logic gates and complex circuits onto a single chip. This was a major breakthrough in the development of integrated circuits.

•**Example**: Microprocessors, small memory chips, and controllers used in embedded systems.

LSI (Large Scale Integration)

- **Transistors per chip:** 1,000 – 10,000
- **Logic gates per chip:** 100s to 1,000s

- **Example:**
  - **IC 8255** – Programmable Peripheral Interface
  - **IC 8085** – 8-bit Microprocessor
  - **IC 2708** – 1 KB EPROM

➤ Used in microprocessors, memory, and I/O controllers.

VLSI (Very Large Scale Integration):

•**Description**: VLSI refers to the integration of **millions of transistors** on a single chip, enabling the creation of highly complex circuits, such as processors, and high-performance computing systems.

•**Example**:

Modern processors like

- Intel Core i7,
- ARM-based processors,
- memory chips, and
- digital signal processors (DSPs).

Design Steps in IC Development

Step-1: Specification

•Define what the IC is supposed to do.
•Includes input/output signals, functionality, speed, power, and size requirements.

Example:
Goal: Clearly define what the IC must do.

Design a 2-input AND gate:
•**Inputs:** A, B
•**Output:** Y = A AND B
•Must operate at 3.3V, respond within 10ns delay, minimal power usage.

Step-2:Design Entry

- Writing the design using **hardware description languages (HDLs)** like **VHDL** or **Verilog**.
- Can also use schematic entry (graphical way) like **Cadence**

Example:

Goal: Describe the design using HDL like VHDL or Verilog.

- **Very High-Speed Integrated Circuit Hardware Description Language**.

VHDL Example:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity AND_Gate is
    Port ( A, B : in STD_LOGIC;
           Y    : out STD_LOGIC);
end AND_Gate;

architecture Behavioral of AND_Gate is
begin
    Y <= A and B;
end Behavioral;
```

Alternate method: You can use **schematic entry** tools like **Xilinx ISE or Quartus** to draw logic gates.

(1) Xilinx ISE Design Suite 14.7 Simulation Tutorial || VHDL Code for AND Gate - YouTube

Step-3. Simulation

- Test the design's logic using a simulator before actual implementation.
- Check if the outputs are correct for different inputs.

Example Simulation

Goal: Check logic functionality with test cases before physical implementation.

- **Example Testbench:**

```
-- A testbench to simulate AND gate behavior
A <= '0'; B <= '0'; -- Y = '0'
A <= '0'; B <= '1'; -- Y = '0'
A <= '1'; B <= '0'; -- Y = '0'
A <= '1'; B <= '1'; -- Y = '1'
```

Tools: ModelSim, Vivado Simulator, etc.

Step-4. Logic Synthesis

•Convert VHDL code into logic gates (AND, OR, NOT).

•Produces a **gate-level netlist**.

Example Logic Synthesis

Goal: Convert the VHDL/Verilog code to a **gate-level netlist**.

- **Output:**

The AND gate is translated to basic logic gate connections using libraries like **NAND2, NOR2** etc., from a target FPGA or ASIC cell library.

- Tool: Xilinx Vivado, Synopsys Design Compiler.

Step-5. Physical Layout

- Map the synthesized design to **physical components**.
- Involves **floorplanning**, **placement**, and **routing** of logic cells on silicon.

Example Physical Layout

Goal: Convert the netlist into a silicon layout.

- Includes:

- **Floorplanning:** Define where blocks will be placed.
- **Placement:** Place gates on chip.
- **Routing:** Connect gates using metal layers.

- **Example Output:** You get a chip layout view with physical paths for A, B inputs and Y output, with power and ground rails.
- Tools: Cadence Virtuoso, Synopsys IC Compiler.

6. Timing Simulation

•Check delays and timing constraints.

•Ensures that signals arrive at the correct time and sequence.

Example: Timing Simulation

Goal: Check **delays**, **setup/hold times**, and **clock timing**.

- **Example:**

Ensure the AND gate responds to input changes within 10ns.

Timing analysis will verify propagation delay from A or B to Y.

- Tools: PrimeTime, Vivado Static Timing Analysis.

Open Source Alternatives to Xilinx Tools:

If you're looking for **open source FPGA tools**, here are options:

- **Yosys – Logic Synthesis Tool**

•Converts Verilog into gate-level netlists

•Open source

- **NextPNR – Placement & Routing**

•Works with several FPGA architectures (like Lattice iCE40, ECP5)

- **Project IceStorm – For Lattice iCE40 FPGAs**

•Fully open source flow from HDL to bitstream

- **SymbiFlow – Aims to create a complete open-source toolchain for multiple FPGA families**

3.6 Electronic Design Automation Tools

1.Capabilities: PCB design, routing, layout automation.
2.Popular Tools: OrCAD, Eagle, Protel.
3.Applications: Accurate and faster PCB and hardware design.

Thank You